The A\* algorithm is a popular pathfinding algorithm used to find the shortest distance between two points and is actually a modification of Dijkstra's algorithm. The A\* algorithm uses a heuristic based on estimating the distance from any searched node to the desired target node, and the A\* algorithm can be made to be Dijkstra's algorithm simply by setting the heuristic cost to zero for all the nodes. The A\* algorithm holds a priority queue of the optimal choices currently held to be possible paths from the origin node to the target node, and when one path 'deeper' in the queue is discovered to be shorter than the shortest path so far, it climbs to the lead of the priority queue. Shortest paths algorithms like A\* are types of combinatorial optimization problems that can be useful for real-world applications like route planning for shipping paths.

Without even parallelizing the code, there are some optimizations that can be made to the A\* algorithm. For example, research has been done discussing the use of precomputed paths for commonly used paths, or using particular landmarks and precalculating paths between them. Additionally, A\* can be modified to be a bidirectional search, wherein two searches are conducted “simultaneously” starting at both the origin node and the target node, branching out until they meet in the middle. The word “simultaneously” is in quotes because while it may appear like the bidirectional search is happening concurrently, with only one thread and/or processor the calculations are still happening serially, alternating between calculating the path from the start node and the path from the target node.

Using multiple threads and altering the algorithm to work more efficiently in a parallel environment, we will show that the parallelized version of the A\* algorithm is more efficient and can be used to compute the shortest path very quickly for large sets of data. For efficiency of runtime, it is important that your graph be connected (i.e., that all your nodes are connected to one another by way of some path in the graph and that no nodes are isolated) or that at least your start node and your end node are part of the same "connected component" of the graph. If your start node and end node are NOT part of the same connected component, your search (bidirectional or otherwise) will search through the ENTIRE search space within the connected component(s) of the graph that has your starting node (and within the full connected component that has your ending node, if doing a bidirectional search) - a very costly operation.

Resources

<--THE RESOURCE DIRECTLY BELOW WAS NOT USED IN WHAT I'VE DONE SO FAR, BUT IT IS PROPERLY FORMATTED

FOR A PAPER THAT RESULTED DUE TO A CONFERENCE-->

Cao, Han, Fei Wang, Xin Fang, Hong-lei Tu, & Jun Shi. (2009, May 19-21). OpenMP Parallel Optimal Path Algorithm and its Performance Analysis. World Congress on Software Engineering, 2009. WCSE '09. Paper presented at World Congress on Software Engineering, Xiamen, China (pp. 61-66).

DOI: 10.1109/WCSE.2009.107